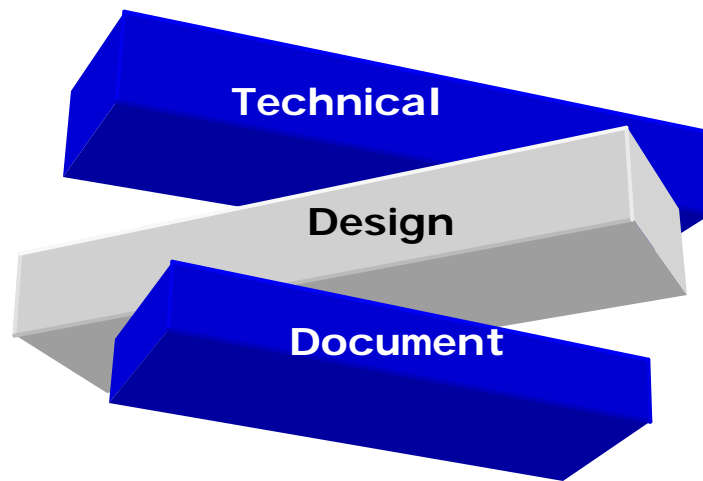


DRAFT

Grid Query



December 26, 2002
Matthew G. Vranicar
Jeremy Simmons
Joshua Gramlich
Rohit Badiyani

DRAFT

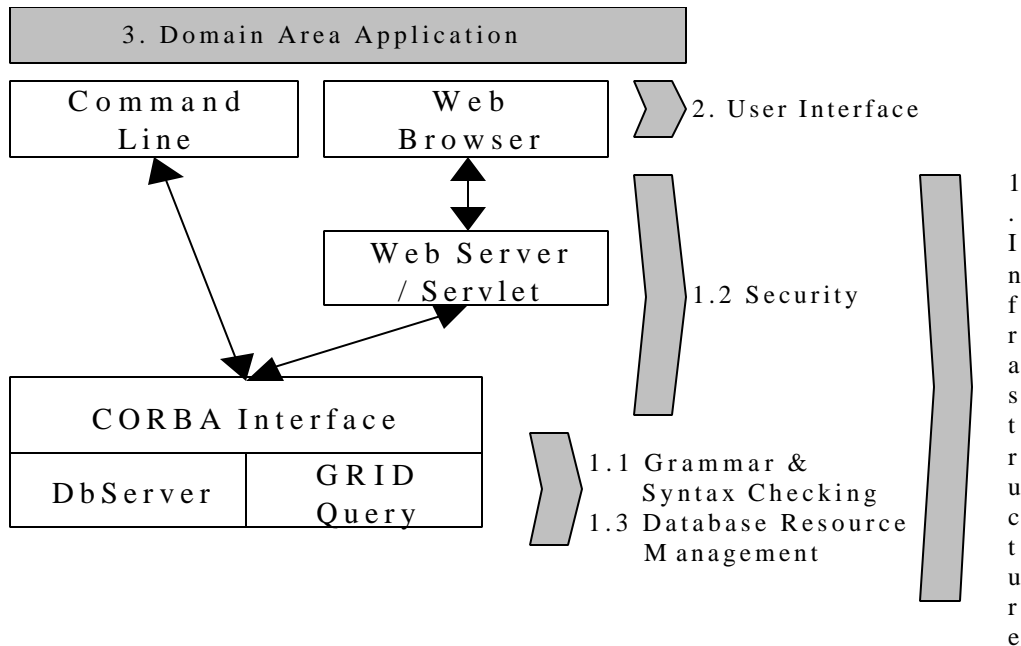
Table of Contents

Components	1
1 Infrastructure	1
1.1 Grammar & Syntax Checking	1
1.1.1 Grammar Parsing	2
1.1.2 Query Building	3
1.1.3 Chain and Link Table Design	4
1.1.4 Phase I Prototype	7
1.2 Security	8
1.2.1 Current Grid Technologies	8
1.2.2 New Grid Technologies	8
1.2.3 Current SAM DB Server Architecture	9
1.2.4 Proposed SAM DB Server Architecture	10
1.2.5 Phase I Prototype	11
1.3 Database Resource Management	11
1.3.1 Query Costing	12
1.3.2 Scan Limits	12
1.3.3 Time Limits	13
1.3.4 UI Impact	13
1.3.5 Test Existing queries	13
1.3.6 Phase I Prototype	14
2 User Interface	15
2.1 Grid Query Extensions	15
2.2 Grid Query Usage in MISWEB	16
2.3 Phase I Prototype	17
3 Domain Area Expertise.....	17
References.....	18
Security	18

DRAFT

Components

The GRID Query project has 3 main components concentrating on Infrastructure, User Interface, and Domain Area Expertise. These three main components of the GRID Query project are described in this document, with full details on their designs. Additional details on existing prototypes developed, working code, and further coding required are included.



GRID Query Project Components
Figure 1

1 Infrastructure

The first effort, Infrastructure makes the GRID Query language more robust, more secure and more capable. It involves changes at all component layers of the GRID Query architecture. With these extensions, the GRID Query language meets the most demanding needs of high volume query users. This effort is actually broken down into three distinct sub-tasks: Grammar and Syntax Checking, Database Resource Management, and Security. Each of these sub-tasks is described individually below.

1.1 Grammar & Syntax Checking

Implementing a more robust, and flexible grammar and syntax checking algorithm requires that certain design goals be met. First, the grammar must be able to be defined in one location, requiring changes to only that location in the code for new grammar conventions to be added, or existing ones changed. Changes can be made to the overall grammar, to suit local needs and conventions, or to otherwise tailor for your installation. The grammar rules must then be read by the parser code, ensuring that user queries are validated against the one version of the grammar rules.

Once the grammar is defined and the parser is built to ensure that all user queries meet this grammar, the next step is to ensure that the output of the parser can feed into the utility which converts it into the required SQL query for the database engine. This requires that the parser output be an agreed upon, understood format, which the SQL Generator utility can use to convert into valid SQL.

DRAFT

To complete the grammar and syntax extensions for the GRID Query capabilities, additional query conditions, not currently allowed are to be added. These include parent/child queries and other more complex conditions. Allowing these queries requires changes to the SQL Generator component, which will provide a new means for defining the new queries. This new means is implemented via a new set of Chain/Link tables which define all the join conditions, and queryable dimensions. These tables eliminate the need for the current Python driver file method used to define the table join conditions. They also allow for more easy addition of new dimensions, since all data required to add a new dimension is now housed in database tables. Dimensions can now be added dynamically, without having to release new versions of software. These chain/link tables are described in detail in section 1.1.3 below.

The components of the new GRID Query Grammar & Syntax Checking and their design benefits are highlighted in the Figure 2, below.

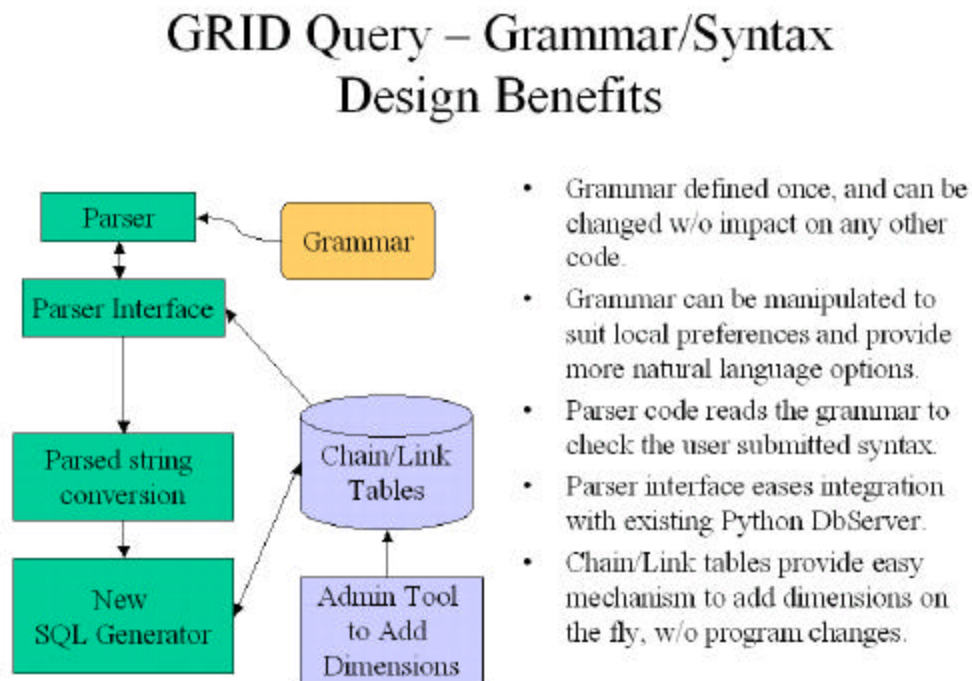


Figure 2

Looking deeper into the components of the design, we can describe it in two main sections, Grammar Parsing, and Query Building. Each of these is described in detail below.

1.1.1 Grammar Parsing

Grammar Parsing requires that the grammar be defined in an acceptable format. The LALR(1) format is chosen for its proven reliability and unambiguity, as well as the availability of tools to read and verify it. In particular, the Lex and Yacc languages work together to provide the necessary parsing and verification tools. The parser built using Lex and Yacc is compiled into a C module, which is then linked as a Python module. This Python module fits perfectly into the current GRID Query architecture, and can be imported directly into the Python based database server code base. This Python library is integrated into the database server code using a parser interface layer, which insulates the Parser and Query Builder layers. The parser interface accepts the output from the parser and reads the chain/link tables only to verify that valid dimension names and data types were provided for the query. A complete picture of the Grammar Parsing layer is shown below in Figure 3.

DRAFT

GRID Query – Grammar/Syntax Design – Grammar Parsing

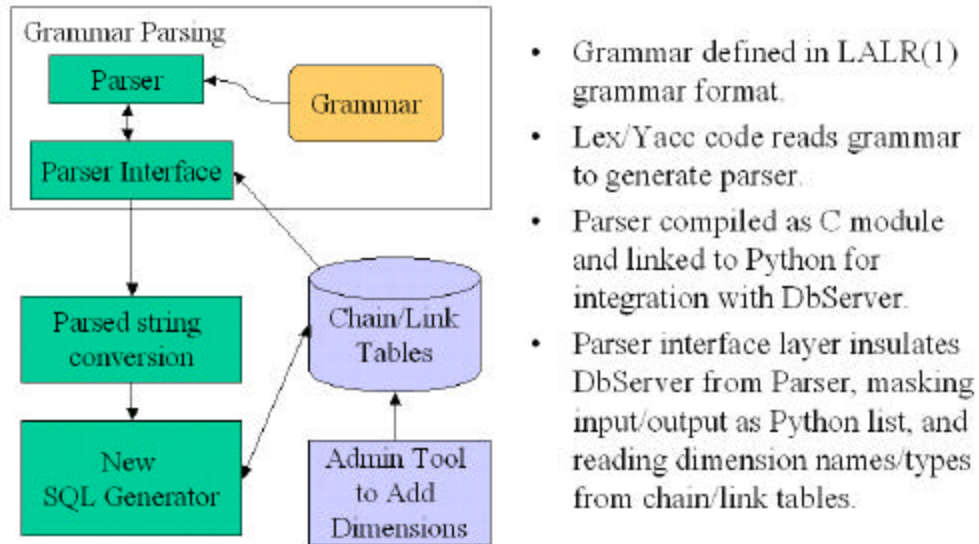


Figure 3

1.1.2 Query Building

The Query Building layer of the parser logic requires that the parser output be read and converted to a valid SQL query. A parser string converter will format the parser output in the proper way for the SQL Generator to understand. The SQL Generator reads the dimension details from the chain/link tables, including all table join conditions. With this data and the dimensions used in the query, the SQL Generator is able to make a valid SQL query.

DRAFT

GRID Query – Grammar/Syntax Design – Query Building

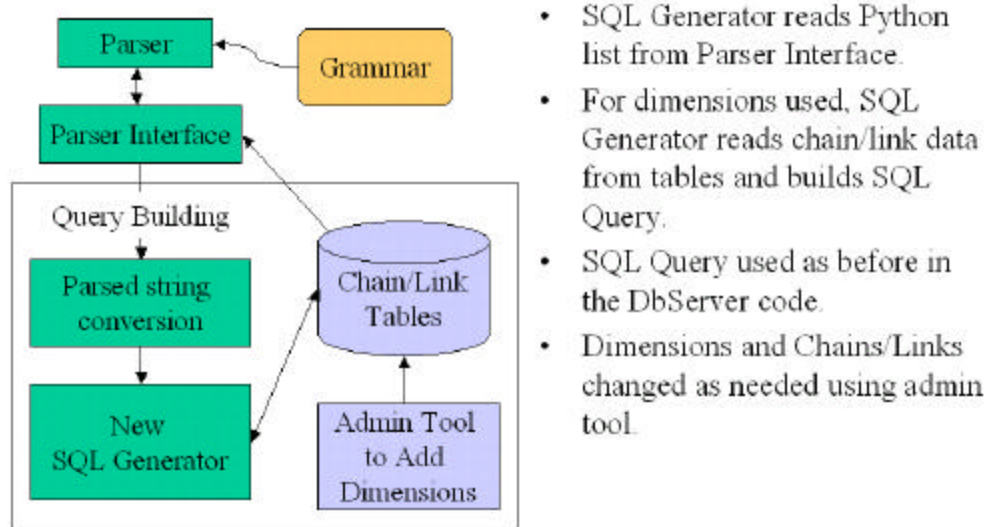


Figure 4

Also shown in Figure 4 above, notice that there is mention of an administrative tool for adding dimensions. This tool allows for the easy entry of additional dimensions, by writing to the Chain/Link tables.

1.1.3 Chain and Link Table Design

The current schema of requiring SAMDbServer code modifications and version releases to add new dimensions for user queries is costing too great an impact on the flexibility of SAM to adapt to user needs. As such, a means for allowing SAM Admin'ers and SAM Shifters to easily add dimensions without modifying the sam_db_server code base is required. This mechanism will store the dimension join details currently found in the DimDriver files into the Oracle database tables. It requires new tables and changes to existing tables to add this ability. The approach, its benefits, and the implementation details of this approach are presented in this section.

Dimensions

The dimension table will lose three columns in this new database schema: DIM_TABLE, DIM_COLUMN, and DIM_ALIAS. These values are moved elsewhere in the new schema as noted below.

Dimension Drivers

The first new concept to introduce is that of the Dimension Drivers. The new table DIMENSION_DRIVERS will house the driver details for using a given Dimension to return facts from a certain table. This replaces the current mechanism of the two different "Driver" files used in SAM, DimDriverDatafile.py and DimDriverDatasetDef.py.

This table has a multi-part key made up of the dimension name, the fact type, and a chain number. The same dimension name can be used to retrieve different facts from two different tables, thus the need to include the fact type. Most dimension/fact combinations will have only one record in this table, and so that record will have a chain number of 1. However, the data model allows the ability to have more than one

DRAFT

chain required to resolve a specific dimension. This will be used to replace the old method of keeping additional required dimensions in the table Dimension_Addons, as was used for the param_category/param_type queries generated for dimensions such as PYTHIA.TOPMASS. There will be two driver records for such dimensions in this model, with one linking its chain back to the table PARAM_CATEGORIES, and one linking its chain back to PARAM_TYPES.

The Answer table and columns represent the table and column names in the end resulting answer to return, or the facts to return from the query.

The Question table and columns represent the table and column names that house the user is trying to use to query the database.

There is also an additional Question Check field, which is actually used in conjunction with the approach mentioned above for housing PARAM_CATEGORIES and PARAM_TYPES. Since dimensions such as PYTHIA.TOPMASS require specific values for the param category and type, a means to include that Check in the dimension definition is needed. This will be accomplished by storing the SQL check required in the Question_Check column, e.g. “= ‘pythia’”, “= ‘topmass’”, etc. Note that this SQL check can be as complex as a nested sub-query, or as simple as the equal to checks noted above. Most current uses of this will be direct equal checks for the MC param/type/category options.

Dimension Depends

The DIMENSION_DEPENDS table houses records indicating which dimensions must be provided by a user if they specify a certain dimension in their query. For example, for performance reasons, we know that they must provide a Run Number when querying Events to ensure that the Oracle database perform an operable, efficient query. This table defines which dependencies are needed for each dimension/driver combination.

Chains and Chain Links

Chains are a tool used to describe the query join paths. A chain starts with one dimension/fact type and depicts a list of links (in the CHAIN_LINKS table) that specify the entire join path all the way back to the fact destination table.

Chain_Name – An arbitrary, unique name of the chain. This name may end up being used to differentiate the type of query you want to perform. For example, for the Child chain noted in the table of example chains below, we may allow syntax such as the following, which somehow triggers the query logic to search for all files where there is a child file that matches the noted file_name pattern.

Child.file_name like ‘%ttbar%muon%’

Links

The Links table depicts the actual join details needed to follow the chains. It includes the following columns:

Link_Name – A unique link name, used to join from the names found in the Chains.Links column back to this Links table.

From_Table – The table on one side (the “from” side) of the link.

To_Table – The table on the other side (the “to” side) of the link.

Link_Columns – The column(s) to be used in the join. The columns are stored as either simply the column name if the same column name is used in both tables. Or, if the names are different, the column is a colon

DRAFT

separated list of columns, with the from table column first and the to table column second, e.g. FILE_ID:SOURCE_FILE_ID from the link from table DATA_FILES to FILE_LINEAGES.

If there are multiple columns required for the join, then the Link_Columns will contain a comma separated list of the multiple key columns. In the case when the column names are identical in both tables, the link columns will look like this example: KEY1,KEY2. In the case when the column names are different in both tables, the link columns will look like this example:
TAB1_KEY1:TAB2_KEY1,TAB1_KEY2:TAB2_KEY2.

Outer joins can also be specified, simply by including a + on the approach key/column combination for the outer join.

When a query is resolved from the links, the From_Table, To_Table and Link_Columns are used

Sample data for the Links table is included in table 1 below.

Link_Name	From_Table	To_Table	Link_Columns
Data_Tier	Data_Tiers	Data_Files	Data_Tier
Logical_Stream	Logical_Streams	Physical_Streams	Logstream_Id
Physical_Stream	Physical_Streams	Data_Files	PhysStream_Id
Child_File	Data_Files	File_Lineages	File_id:Source_File_Id
Child_Lineage	File_Lineages	Data_Files	Dest_File_Id:File_Id

Sample Links Data
Table 1

In addition to gaining the ability to add dimensions, even those that include new tables in the resulting query, we gain something else with this feature. We gain the ability to easily check the join paths every time a new dimension is added. We simply walk the paths of all Links on the Chains described below and we can ensure that new table additions will indeed give proper query results when used. This is currently not the case when editing the DimDriver files. There are no checks that the python dictionary changes a SAM Admin'er makes will work properly. In fact, we have seen situations recently where errant addition of a new condition caused the query join logic to write a query without all the proper joins in place.

SAM_DB Impact Analysis

Here are the steps, including sequence, in which the changes for this work must be introduced into the SAM environment.

- 1.) Add the new tables DIMENSION_DRIVERS, CHAINS, LINKS, CHAIN_LINKS.
- 2.) Modify the table DIMENSION_DEPENDS = Add columns DIM_FACT_TYPE and CHAIN_NUMBER, and include them as part of the primary key, along with DIMENSION_NAME and REQUIRED_DIMENSION_NAME.
- 3.) Add the code changes to use these new structures in the SQL Generator.
- 4.) Drop the columns DIM_ALIAS, DIM_TABLE, DIM_COLUMN from the table DIMENSIONS.

DRAFT

1.1.4 Phase I Prototype

In Phase I of the GRID Query SBIR project, a prototype of the design outlined above was built for the Grammar & Syntax Checking. This prototype which included all the required parser logic, but relied on the original, current SQL Generator. Also, the new Chain/Link tables and the administrative tools to maintain them were not built. Figure 5 below depicts the functionality built in the prototype.

The Grammar definition and parsing of it using the Lex/Yacc tools is proven to work, but not all components of the grammar were included in this working sample. Also, the parser interface now reads all dimension names and types directly from the Dimensions table, since the new Chain/Link tables are not yet available. Finally, after the query is parsed, the parser output is converted to a format compatible with the current SQL Generator utility, which uses the same Python DimDriver style files to derive the table joins and not the new Chain/Link tables.

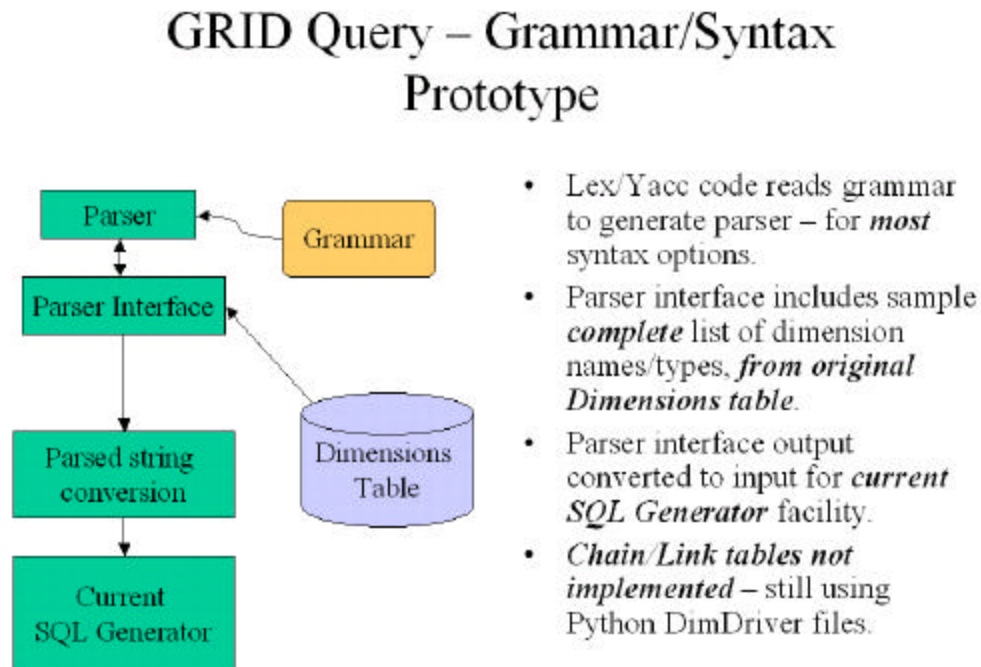


Figure 5

The current prototype works as is for a few simple examples. Or, alternatively, it can be effectively turned on or off, by simply setting one flag variable in the new file, DimParser.py.

Moving forward, the following list of features must be built to provide all of the design benefits of the new Grammar & Syntax Checking capabilities.

- Improve mechanism for feeding updated dimension names to parser. Currently the code is reading from the database for every query. This would ideally be done once by the Parser Interface layer and kept in the database server memory.
- Add missing options to syntax rules, such as date and datetime dimensions. At least all syntax that is currently allowed must be built. Additional syntax options can also be considered.
- Perform thoroughness check of syntax, to ensure that all the existing working queries work when converted to the new code base.
- Create grammar documentation, which can be read by the non-technical end users of the GRID Query system.

DRAFT

- Build and populate chain/link tables in the development database, and include a conversion script for migrating this to the integration and production databases.
- Modify SQL Generator code to use the new chain/link method to create the SQL Query.
- Test on existing dataset queries. Either all, or at least a broad enough subset of dataset queries must be executed using the new GRID Query engine to ensure that they work properly.
- Modify grammar options, adding new conventions now allowable given the new Chain/Link join definition format, such as “parent.”, “child.”, “raw”, “production” etc.
- Include detailed documentation in all programs, to facilitate code maintenance.

1.2 Security

When SAM was first created, security and resource management were not the first priority for the developers of the SAM Db Server. The main goal was to make a system that physicists could use to easily, and quickly find their data. But, as the use of SAM has expanded to reach a worldwide audience, concerns about security and resource management are now prominent. In addition, much ado has been made over recent developments in Grid technology. Grid services are gaining more and more traction within the high energy physics community, and now, SAM DB Server is now being considered for its part in the Particle Physics Data Grid (<http://www.ppdg.net>). In order for the SAM DB Server to be compatible with true Grid services, security and resource management must be integrated into the existing framework.

1.2.1 Current Grid Technologies

The Globus Toolkit (<http://www.globus.org>) is one of the most popular methods of securing resources while making them available to large scale computing Grids. SAM-Grid is one current project at FNAL that uses the Globus toolkit [GLOBUS] (as well as Condor, www.cs.wisc.edu/condor/) to Grid-ify SAM client requests. SAM jobs are submitted via SAM-Grid, and SAM-Grid determines which particular resource on the Grid is suited to the task, and assigns the job to that resource.

Unlike SAM-Grid, the SAM DB Server has no current mechanisms that will allow for security, authentication, and credential delegation between the various locations in which SAM is being used around the world. Like SAM-Grid, we plan to use the Globus toolkit in our design making the SAM DB Server “Grid aware”. [SAM-GRID]

One technology not specifically handled by the current incarnation of the Globus toolkit is object handling. SAM DB Server currently uses CORBA to provide object brokering and handling, but limitations between CORBA and the Globus toolkit will not provide SAM DB Server with the required level of security. In order to satisfy the security requirements of FNAL we must look beyond CORBA towards Web Services. Web Services are central to the technology being implemented in the next version of the Globus toolkit.

1.2.2 New Grid Technologies

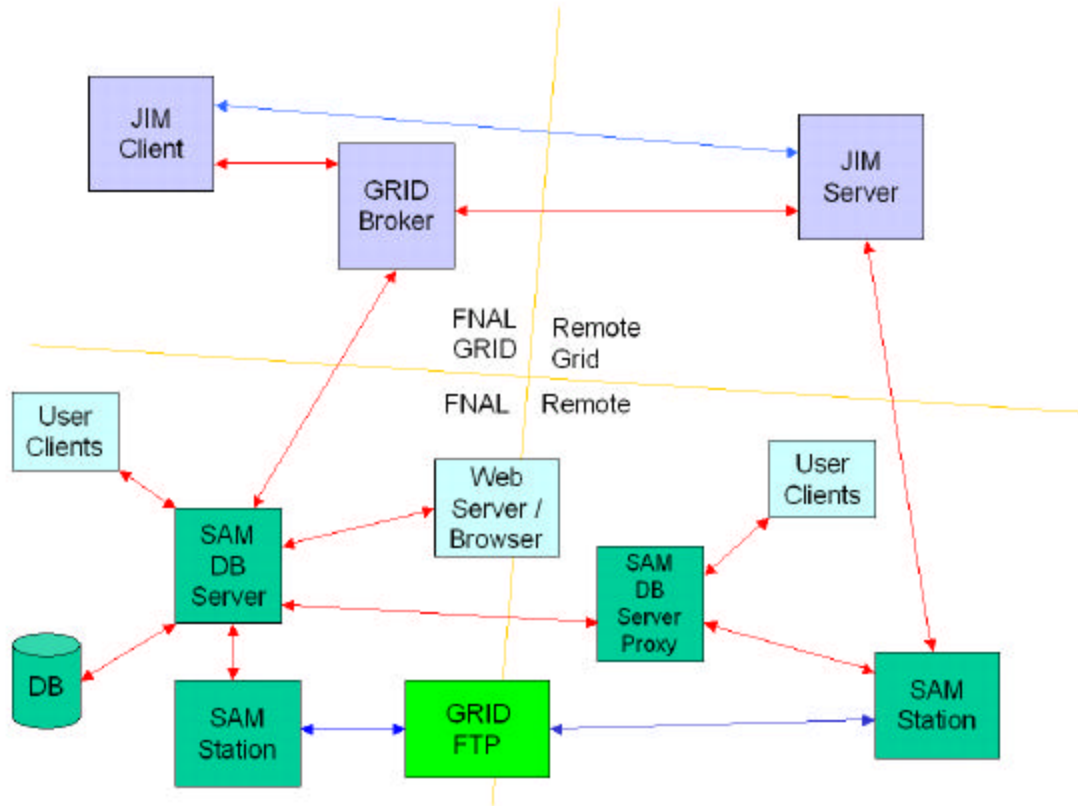
The Globus toolkit version 3.0, or OGSA specification, is based in part upon the WS-Security specification developed by IBM, Microsoft and Verisign. In order to take advantage of the security services offered by OGSA and WS-Security, current communication mechanisms within the SAM DB Server that are provided by CORBA should be converted to or replaced with WS-Security/OGSA mechanisms. A Web Services object that handles the security and communication between client and server will provide the necessary security requirements, such as authorization, encryption, delegation and privacy, as well as the ability to use heterogeneous client and server software, to SAM DB Server.

In using Web Services, the fundamental architecture of the SAM DB Server should not change. The biggest difference is the certificate mechanisms that will now have to be included in the server and clients. Each client must be issued a certificate from a recognized (by FNAL) certificate authority (very likely to be FNAL). Also, each server/service must be issued a certificate as Globus security methods depend upon mutual authentication via these certificates.

DRAFT

1.2.3 Current SAM DB Server Architecture

Figure 6 represents the different pieces of SAM as it stands today. The blue connection identifiers connecting the JIM Client to JIM Server, the SAM Station to GRID FTP, and GRID FTP to the SAM Station represent links that are secured in some way, shape or form. The JIM client and server (part of the SAM Grid project) have been Grid enabled with the Globus toolkit, and identify each other through the use of x509 certificates. The SAM Stations use service certificates to identify themselves to the GridFTP service when doing file retrieval.



Current SAM Db Server Architecture

Figure 6

The current configuration of SAM and its constituent systems allows for misuse and misallocation of the SAM services. While communication is protected in some areas, it is left open in others. In order to protect the considerable resources that the DoE and many other organizations have contributed to the SAM project, we must add security measures.

SAM DB Server is a python application that uses a CORBA object broker to facilitate communication between the DB Server and SAM clients. During our investigation of the Globus toolkit and CORBA, we found that integrating CORBA and Globus was not going to solve the problems this project was meant to deal with. Using CORBA and Globus will not give us the levels of utility, that is, the quality of authentication, encryption, delegation, etc., that is desired of SAM DB Server.

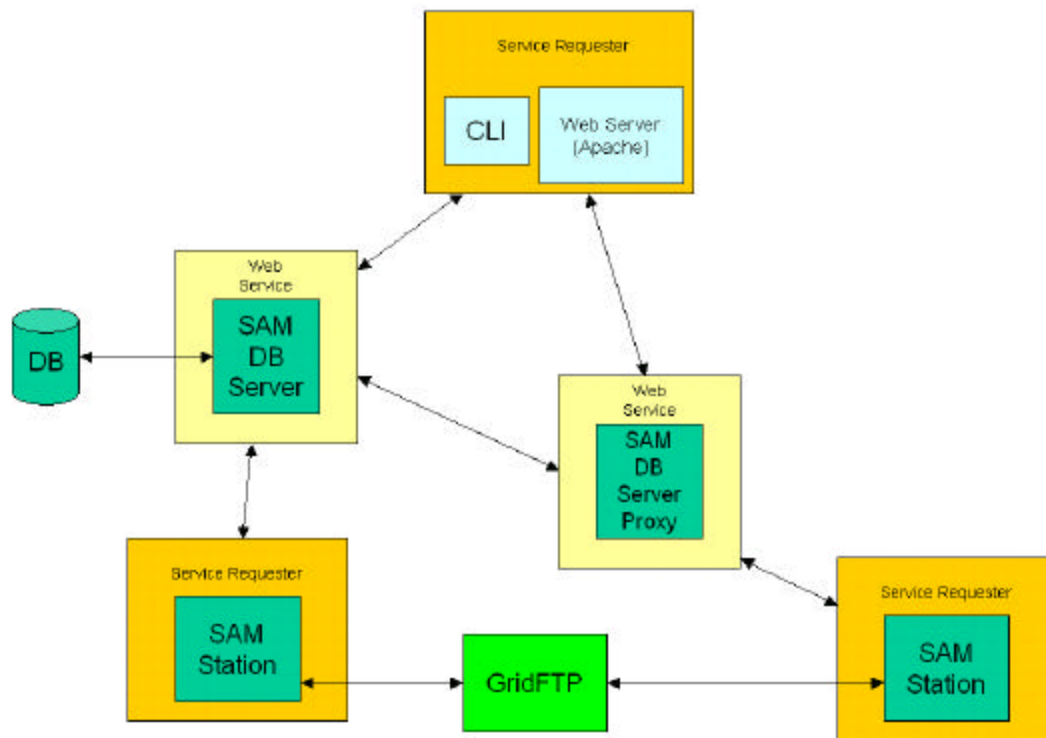
The proposed strategy for Open Grid Services Architecture solves many of the problems with Globus and CORBA interoperability. OGSA provides solutions to the SAM DB Server security issues by using Web Services security mechanisms. [WSS] WS-Security is a specification developed by IBM, Microsoft and Verisign that provides secure communications via Web Services. It contains provisions for passing both x509 certificates and Kerberos tickets in an authentication and encryption scheme.

DRAFT

These measures will be based upon the Open Grid Services Architecture[SAOGS]. OGSA provides us with guidelines for using a security model, methods and components to create a secure Grid environment.

1.2.4 Proposed SAM DB Server Architecture

Our vision of the future incarnation of SAM and its methods of communication follows diagram 7. In this architecture, Web Services are wrapped around the SAM DB Server instances, and the clients to those services are wrapped in Web Services Requesters. Web Services will allow heterogeneous client and server software pieces to speak with each other. Web Services will also enable the type of security methods we would like to use. That is, the new Globus toolkit version 3.0 will be integrated with security methods similar to the current WS-Security specification. For all clients, there will be a single interface, through which messages, certificates and proxy delegation will be handled.



Future SAM Db Server Architecture
Figure 7

It should be known that the technology surrounding this proposal is at current, unfinished. Much noted here depends upon the completion of the Globus 3.0 toolkit, and the use of OGSA tools based upon the WS-Security specification. The current incarnation of the WS-Security specification does not meet all the needs of OGSA. That functionality will be added to the Globus toolkit 3.0 to meet all the security requirements of OGSA [OGSA-RM]. The Globus toolkit version 3.0 is scheduled to debut in alpha in January of 2003.

The Globus toolkit gives us security based upon certificates. These certificates are managed by a certificate authority (CA). Any organization can act as a CA by operating a CA server. It is up to the receiving end to determine whether or not the issuing CA is to be trusted. Once a trust and identity have been established, the user and service may interact.

DRAFT

Before a user can communicate with a service or another user with his or her certificate, a proxy certificate must be generated. With Globus, the command is “grid-proxy-init”. The command uses the certificate issued by the CA to generate a proxy certificate. The proxy certificate is generated with the original certificate. The proxy certificate is then transmitted to the service for authentication and authorization.

The service receives the proxy certificate and then compares it to the certificate held by the CA. The CA then can determine whether or not the proxy certificate is valid, because only an original copy of the certificate may decrypt the proxy. If the proxy can be decrypted by the copy of the user certificate held by the CA, then it is determined to be valid. At this point, the service would recognize the user and apply whatever local authority that has been given to the identified user.

The proxy certificate may also be used in delegation of authority. If the service that a user has requested services from must go to a third (or fourth, fifth, etc.) service to obtain resources, the first service may use the proxy certificate of the original user as its credentials. In this scenario, then the user is requesting the resources, and can be individually identified as opposed to the first service being the requester.

The way this would apply to SAM is in the use of the SAM Grid. A JIM client would send a request to a JIM server and be authenticated. Part of that client request is a request for files. The JIM server would then request file locations from the SAM DB Server using the delegated credentials of the proxy certificate. This way, the SAM DB Server is processing the request for a particular user.

The JIM server would then use the delegated credentials against the SAM Stations for file retrieval. The benefits of this arrangement allow each resource to individually identify the resource requester. Such an audit trail is an important piece of security management in any system.

1.2.5 Phase I Prototype

During Phase I, the security effort has involved more investigating and sampling, then actually developing of a prototype. PIOCON has made the current SAM GRID architecture work across machines in its office, in an effort to understand and prove the current recommended alternatives. PIOCON has also investigated much of the research and document by other key GRID players, to understand the future direction of security in the GRID realm.

Moving forward into Phase II, the Security architecture described above will be built. But, this architecture is still open to flexibility, and will be further refined as needed to include the emerging GRID solutions.

1.3 Database Resource Management

The Database Resource Management components allows restrictions on database resource utilization by individual and group users. Resource “Levels” control the highest amount of database optimization “score” that a user is allowed in their queries, along with a limit on the table scanning that can be performed. The ability to limit query execution to a given amount of time is also allowed. Below is a summary of the design benefits of the Database Resource Management component of the GRID Query project. They are explained in detail below.

Design Benefits & Details

- Query Costing
 - User “Group” or “Level” query limits
 - Analyzing query cost before executing
- Scan Limits
 - Table based scan limits
- Time Limits
 - Query execution time limits

DRAFT

- User Interface Impact
 - Minimal impact on user interface
 - User interface backward compatibility
- Test Existing queries

1.3.1 Query Costing

User “Group” or “Level” query limits are enforced using the following techniques. Each user is assigned to one or more resource management groups. These groups will reflect user roles, such as General User, Shifter, Expert, Administrator, etc. In turn, each of these roles is assigned to a particular resource management level. This level controls all of the resource management constraints. The following additional design factors are enforced for this level.

- Default assigned user query level: Each role has a default assigned query level for users in that role.
- Maximum override user query level: Each role also has a maximum allowable override level. This override level allows the users to execute queries at a level higher than their normal level, but only if they know the proper syntax for indicating the higher level.
- Flexible number of levels (1 – 5): A flexible number of resource usage levels allows this design factor to be adjusted as necessary. Currently, five levels are considered adequate.
- *NOTE: Prior to security additions to the DbServer, which will allow us to differentiate users, all users will operate at same level.*

Analyzing query cost before executing is performed for every query executed through the Grid Query mechanism. The SQL cost is analyzed and compared to the allowable cost for the user’s resource usage level to determine if the query may be executed. The components of this design factor include the following.

- Analyze Oracle Optimizer results for cost: The Oracle Optimizer cost is derived for the SQL statement, using the built-in cost based optimizer.
- Compare costs of query to assigned user level: The optimizer cost is compared to the allowable cost of the user’s resource usage level. If the cost is less than or equal to their allowable cost, then the query will be executed. If the cost is greater than their allowable cost, they will receive a “query resource usage limit exceeded” type error message.
- Dynamically modify cost levels as database grows and optimizer cost factors change: A method exists in the DbServer that can be used to dynamically calculate the optimizer costs for various queries. From this method, statistics can be analyzed to properly set the resource usage level allowable costs.

1.3.2 Scan Limits

Table based scan limits are enforced at the Grid Query user query resource level. Each level may have one or more tables which it is prohibited to scan. Thus, if a user is operating at a query level below or equal to a query level for which a table scan is prohibited, any SQL query that tries to scan that table is not allowed. See the examples below, based on the table below, for clarification.

DRAFT

Example 1: User Joe is not allowed to execute a query that scans table Data_Files, but all other users may.

Example 2: Users Joe and Sue are not allowed to execute a query that scans table Events, but others may.

Example 3: User Sammy is the only one allowed to execute a query that scans table Event_Catalogs.

User	Role	Query Level	Scan Limits
Joe	General User	1	Data_Files
Sue	Expert	3	Events
Sally	Shifter	4	Event_Catalogs
Sammy	Admin	5	

Sample Query Levels and Scan Limits

Table 2

1.3.3 Time Limits

Query execution time limits are enforced for all queries by level. Each level has an allowable timeout threshold, after which the query is terminated and an error returned. This requires that queries execute in threads, which may be cancelled upon reaching given time thresholds.

NOTE: This feature depends on the multi-threading capabilities of the DbServer product into which the Grid Query mechanism is used. If this multi-threading is not available, then this time limit feature is not.

1.3.4 UI Impact

Minimal impact on user interface is an essential requirement in the implementation of the database resource management component. Since multiple clients are already deployed, this is essential. The following factors are vital for this database resource management feature to be successfully deployed in clients that rely on the Grid Meta-Data Query utility.

- Current interfaces run as is, with all queries run under the users default limit: Without changing any interfaces, the queries will run as is, with an agreed upon default limit assigned to each user. This limit may be “unlimited”, or it may impose a stricter to eliminate the run-away query troubles that currently exist.
- User can exceed their default limit, but only up-to their allowable maximum override limit: When users run a query, they can exceed their default limit, but only up-to their allowable maximum override limit. They accomplish this using the new limit option noted below, which will be allowed in both command line and web query interfaces.
- New option in Sam User Command-line-interface
 - --limit = n
- New option in Sam Project Editor Java Servlet
 - New web field for level parameter input
- For backward compatibility, the default limits can be set to maximum, in effect no limit.

1.3.5 Test Existing queries

Any queries that have been used to create existing datasets may be tested using the new costDatasetQueries method. This method relies on the fact that the SQL query executed for each dataset is saved the moment that the dataset is captured. It allows for all dataset queries to be tested, or a subset of the queries based on a number of parameters. The method and its input parameters are denoted below.

Note, that if you decide to calculate the execution time as well as the optimizer cost, then you may incur significant waits and may impede user performance, as the only way to calculate this time is to actually run each query. Before using this option, make sure you consult with your DbServer administrator.

- void costDatasetQueries (
in long rows, /* number of dataset queries to test */

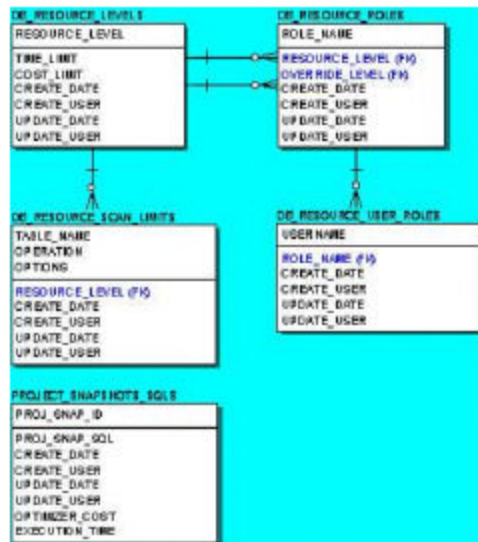
DRAFT

```
        in string since,          /* test all queries created since this date & time */
        in string until,         /* test all queries created up until this date & time */
        in boolean calctime      /* calculate the query execution time too? */
    )
    raises(DimensionError, DBError, InternalException);
```

Upon completion, this method writes the optimizer costs, and time if calculated, into the existing DATASETS_SQL table, alongside the SQL query. This cost can then be analyzed and compared via histograms or other means to derive proper cost ranges for the various database resource management usage levels.

This method will want to be run once, but may also be run as often as needed, on as big or little a sub-set of the datasets as required.

GRID Query – Resource Management Design – Database Tables



- Resource Levels
 - 1 to 5, w/ cost & time limits
- Resource Scan Limits
 - Tables and scan types limited by level
- Resource Roles
 - Named roles w/ assigned levels and allowable overrides
- Resource User Roles
 - Assignments of Users to Roles
- Project Snapshots SQLs
 - Records cost and execution time when Snapshot created

Figure 8

1.3.6 Phase I Prototype

The current database resource management code is working and will be introduced into the SAMDbServer environment shortly. The following simple tasks are the only issues required to implement this component of the Grid Meta-Data Query Language into a production DbServer:

- Deploy tables stored in grid_query/sql in the sam_db schema
- Release grid_query version 1.0
- Release sam_db_server from CVS branch (sbir1)

Moving forward, the following issues remain in implementing the complete design of Database Resource Management as outlined above:

- Query Execution Time Limits
 - Current single-threaded server does not support this.

DRAFT

- Must multi-thread server to add this ability.
- To tie execution time limits to user query level requires user specific security in the DbServer.
- Defining Limits
 - Run cost calculations and analyze optimizer costs to calculate proper levels and limits.
 - Based on cost analysis, determine table scan limits.
 - Ability to dynamically adjust limits might not be desirable – this may always want to be a task requiring analysis.
- User Level definitions
 - Interfaces are needed for adding/modifying levels and limits.
- Integrate with New DAN Architecture
 - Switch from DbCore to Sql/Identity method for DAN Server with Multi-Threaded OmniORB architecture.
 - Assuming Multi-Threaded model to implement user specific database resource usage limits
- Integrate with Security implementation
 - Creation of CORBA Servant will identify the client user to the server code, by passing site specific credentials, either operating system user or security principal.
 - Assume identification of security context upon Servant creation is enough, without requiring passing of security context with each CORBA interface.
 - Non-authenticated users provided base level limits.

2 User Interface

Currently, SAM database queries are executed in a number of distinct ways. The dataset definition queries are executed using the dimensional query logic of this Grid Meta-Data Query Language. Both the sam_user command line interface and the sam_project_editor web interface allow queries to be specified in dimension name and constraint value pairs. The sam_data_browsing product, on the other hand, uses the old MISWEB technology (<http://miscomp.fnal.gov/misweb>) to deploy SAM database queries.

The goal of this effort is to integrate the Grid Meta-Data Query Language into the MISWEB query pages. This will allow the numerous html based query pages to rely on the same dimensional query logic as the dataset definition queries. Since the MISWEB pages require database specific logic, such as table names, column names, where clauses, etc., this eliminates the need to maintain the query pages as database constructs are changed, added, or removed.

The design benefits of this effort are outlined here:

- Combine multiple disparate query tools into one
 - Dataset definition queries w/ dimension logic
 - SAM Data Browsing w/ html based MISWEB logic
- Get consistent, predictable results from queries
 - where ever executed: command line interface, web dataset editor, web query
 - esp. for data files based on availability, location, etc.
- Eliminate SQL “smarts” behind SAM Data Browsing pages: table names, column names, etc.
- Leverage database resource management, security, and other options of Grid Meta-Data Query.

2.1 Grid Query Extensions

In order to accomplish these design benefits, the first step is to make the Grid Meta-Data SQL query generation utility built into the SAM DbServer available to external programs such as MISWEB. This requires the addition of a new method in the DbServer that accepts dimension query input and returns a SQL statement as output. This method is depicted below.

```
string queryDimensionsSQL (
    in DimensionValue dims, in long limit )
```

DRAFT

```
raises(DimensionError, DBError,  
       InternalException);
```

To fully extend the functionality of Grid Query out to all MISWEB query pages, additional input is also required to the above method, and must be accepted by the Grid Query logic when building the dimension based query. The following options must be allowed as input to provide ultimate flexibility in query derivation. They must be added to the method noted above.

Additions Required to Grid Query Logic

- Facts – the facts to return from the query are specified, mapping back to database columns.
- Metrics – metrics allow for functions to be performed on various facts.
- Orders – indicate the sequence for order by clauses.
- Groups – indicate the groupings to use for aggregate functions.

Adding these new options may even require the addition of another table to the Grammar and Syntax checking details of this overall Grid Query product design. A Fact table might be necessary to control which facts are allowable as return values from which tables, and what metrics can be applied to which facts.

2.2 Grid Query Usage in MISWEB

The next step is to integrate the Grid Query logic into MISWEB query pages. This is a tricky business, as we really do not want to rewrite every MISWEB query page. Instead, we must attempt to minimize the impact of change on the existing html based query pages. Thus far, we have come up with the need for the following changes to existing pages if they are to take advantage of the Grid Query logic.

Basic Changes Required for MISWEB and Grid Query Integration

- Add flag indicating dimension query logic, e.g. dim_flag = 1
- Change “wheres” to dimensions, e.g. where_columnN becomes wheres_dimN
- Change “columns” to facts/metrics, e.g. columns becomes facts

With these minimal changes, the underlying MISWEB code will be modified to call the Grid Query logic when executing a query that is flagged as a dimension query. Additionally, there may be a need to include further logic that MISWEB pages can pass as add-on logic to the dimension query. The SAM DbServer uses this trick internally, and so MISWEB query pages must be analyzed to review their need for similar functionality.

Other features can also be enabled, such as the Database Resource Management features noted earlier in this document. A limit indicator might be added to MISWEB query pages to control the user override of their default limit.

Additionally, the current MISWEB options to either Run, Build, Edit, or Make a query must be reviewed. The prototype effort concentrates solely on Run, but the end product must provide for the abilities of the Build, Edit, and Make options if it is to replace the current MISWEB functionality.

2.3 Phase I Prototype

The Phase I Prototype implements the basic new queryDimensionSQL method above, without the full implementation of the facts, metrics, orders, and groups. This method is available as a SAM DbServer method and may be used from any SAM client.

On the MISWEB query usage side, the first step is to modify MISWEB to allow it to call Grid Query to resolve the SQL query, instead of using its own internal logic. Also, a few sample MISWEB html query pages are built which show the minimal impact required to change these pages to use the new query logic.

DRAFT

These sample query pages provide extremely limited query functionality only, as they use only the currently available dimension queries, and the currently available dimension query return facts. They are intended as a proof of concept prototype only, and currently query only the data_files and dataset_definitions tables.

Moving forward, the following issues remain in implementing the complete design of User Interface Extensions as outlined above:

- Add all SAM tables as “driver” tables for dimension queries.
 - New Chain/Link design provides this (see Grammar/Syntax).
- Add new Facts, Metrics, Functions, Orders, Groups options.
 - MISWEB flexibility requires the same capability in the new SAM DbServer method.
- Integrate Db Resource Management Limits.
 - Integrate limits into the MISWEB html query pages, by adding another Misweb parameter.
 - Build resource management error handling into MISWEB.
- Integrate with Grid security, or else just run as “read-only” user.
- Figure out how MISWEB Design/Build/Make options interact with Grid Query Logic.

3 Domain Area Expertise

Change existing interfaces and add new interfaces, better applying them to the High Energy Physics community’s needs for the SAM project.

DRAFT

References

Security

[SAOGS] The Security Architecture for Open Grid Services, N. Nagaratnam, P. Janson, J. Dayka, A. Nadalin, F. Siebenlist, V. Welch, I. Foster, S. Tuecke, *The Globus Project – OGSA Security*, July 2002. <http://www.globus.org/ogsa/Security/>

[WSS] Web Services Security, B. Atkinson, G. Della-Libera, S. Hada, M. Hondo, P. Hallam-Baker, J. Klein, B. LaMacchia, P. Leach, J. Manfredelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, D. Simon, *IBM DeveloperWorks*, April 2002.

[OGSA-RM] OGSA Security Roadmap, F. Siebenlist, V. Welch, S. Tuecke, I. Foster, N. Nagarantnam, P. Janson, J. Dayka, A. Nadalin, <http://www.globus.org/ogsa/Security>, July 2002.

[GLOBUS] The Anatomy of the Grid, I. Foster, C. Kesselman, S. Tuecke, *International J. Supercomputer Applications*, 15(3), 2001.

[SAM-GRID] SAM-Grid: Using SAM and Grid Middleware to Enable Full Function Grid Computing, A. Baranovsk, I. Bertram, G. Garzoglio, L. Lueking, I. Terekhov, R. Walker, *Beauty* 2002, July 2002.

Matthew G. Vranicar is a Partner with PIOCON Technologies, a Chicago-based Data Intelligence consulting firm.

Jeremy Simmons is a Principal Consultant with PIOCON Technologies.

Joshua Gramlich and Rohit Badiyani are Consultants with PIOCON Technologies.

Questions or comments about this article can be directed to vrnicar@piocon.com. Please visit www.piocon.com for detailed information regarding PIOCON Technologies offerings and services.